APPLICATION FOR
UNITED STATES PATENT
IN THE NAME OF

SOHEILA ABDOLBAGHIAN AND TUAN NGUYEN

ASSIGNED TO

TOSHIBA AMERICA INFORMATION SYSTEMS, INC.

FOR

# SYSTEM AND METHOD FOR COMPUTER APPLICATION MANAGEMENT

DOCKET NO. 81087-250435

PREPARED BY:
PILLSBURY MADISON & SUTRO LLP
725 S. FIGUEROA STREET, SUITE 1200
LOS ANGELES, CA 90017
(213) 488-7100

EXPRESS MAIL NO. EL 331 915 435 US

# WEB CLIENT SESSION MANAGEMENT

## BACKGROUND

In general, a computer executing an application may terminate the application unless the computer senses a timeout period restart event, such as a keystroke, a mouse button click, pointer motion, touch screen activation, message receipt, ping signal receipt, or a like input. Reasons for terminating applications based on the absence of a timeout period restart event include conservation of RAM cache and memory, power and other computer resources,

In a communications network with a client/connected computer architecture, communication between a client and a connected computer occurs in sessions, each of which is generally given a unique session identifier (SID). Each session may be associated with a unique client identifier or address (e.g., an Internet protocol address). During a session, the connected computer may receive from the client multiple service requests, each of which may relate to a number of applications. Applications, particularly those that utilize a large amount of connected computer or network resources (e.g., those that are memory-intensive), may have a timeout feature that terminates the application in a session if an application event is not sensed or a response to an information request is not received within a specified period of time.

Where a session includes multiple applications, a user operating on a client terminal may not respond to an information request or cause an application event because the application has been moved to the background of the client terminal and the user is actively working with a second application in the client terminal foreground. It is often undesirable to have the first application timeout in such situations.

Moreover, some applications prompt the user for a specified input prior to timing out the application. For example, applications that bill the user based on usage time often cause the client terminal to display a window asking the user if the user wants to maintain the application connection. The window may include an icon that may be activated to cause the terminal to send

the application a message sufficient to avoid application timeout. Otherwise, application timeout proceeds as scheduled.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 depicts an embodiment of a communication network according to the present invention;

FIGURE 2 is a flowchart illustrating the logical process that may be executed by a timeout function according to an embodiment of the present invention; and

FIGURE 3 is a flowchart illustrating the logical process that may be executed by a keep-alive process according to an embodiment of the present invention.

FIGURE 4 is a flowchart illustrating the logical process that may be executed by a keep-alive process according to another embodiment of the present invention.

DETAILED DESCRIPTION

The present invention is generally directed to a system and method in which a second application (the "keep-alive function") sends a ping, message or other signal to a first application to prevent the first application from being timed out (hereinafter, "keep-alive input"). In an embodiment of the invention, the first application is queried as to the appropriate content and format of a timeout period restart message. The keep-alive function may be automatically executed upon the occurrence of a triggering event. The keep-alive function may, periodically or based on a timeout clock related to the first application, check the status of the first application and/or send a keep-alive input.

FIGURE 1 depicts an embodiment of the system according to the present invention. User terminals **1a-1c** (collectively, "user terminal **1**") used by users **2a-2c** (collectively, "user **2**") may be connected to a communication network **4** so that the user terminal **1** may send signals, data, messages and the like to the connected computers **3a-3c** (collectively, "connected computer **3**") and receive the same from the connected computer **3**. Other terminals may also be connected to the communication network **4**. The user terminal **1** may have an operating system(s) which can be used to execute software applications. The user **2** may cause the user terminal **1** to execute various software applications, including for example, Web browser, word processor, spreadsheet, database, hardware driver, electronic mail, compiler, video or audio data player, customized software, network resource management, or other applications. The operating system for the user terminal **1** may be different from the operating system for the connected computer **3**. Note that although the phrase "terminal" is used to describe the user terminal **1**, the user terminal **1** may include a computer processor that is independent of the connected computer **3**, allowing local execution of applications or portions thereof. The user terminal **1** may run applications either as desktop, foreground or background processes. In particular embodiments of the invention, the user terminal **1** may be connected to other terminals or computers in a local area network (LAN) and may be connected to the communication network **4** through a gateway or firewall.

The communication network **4** may be a local area network (LAN), metropolitan area network (MAN), wide area network (WAN), the Internet or any other type of computer network and may use any network implementation, including, for example, the Ethernet, ARCnet, Token Ring implementations. Information communicated over the communication network **4** may conform to any data communications protocol, including TCP/IP, IPX/SPX, NetBios and AppleTalk. The communication network **4** may include wire line (such as twisted-pair telephone wire, coaxial cable, electric power line, optical fiber wire, leased line or the like) or wireless (such as satellite, cellular, radio frequency or the like) connections.

In an embodiment of the invention, the user **2** may cause the connected computer **3** to execute a first software application using the user terminal **1** and may subsequently cause the connected computer **3** to execute a second software application. One or several copies of the software applications may reside on the connected computer **3**. Alternatively, a software application may consist of discrete parts which are distributed between the connected computer **3** and the user terminal **1**. Multiple copies of a particular software application, portions thereof, or shared or dedicated resources used by the application may be in use by different users simultaneously. The first software application may include a function (the "timeout function") whereby the application is terminated if the user **2** does not take some action or the user terminal **1** does not transmit a signal or message recognizable by the first application within a specified time period (the "timeout period"). This is a particularly useful feature when an application requires the use of shared memory, processor, data source or other resources.

In a particular embodiment of the invention, the connected computer **3** may be a server and the user terminal **1** may establish a client-server session with the connected computer **3** by transmitting a session request to the connected computer **3**. The connected computer **3** may establish a session and return to the user terminal **1** a session identifier. A first application, e.g., the downloading of content associated with a particular Web page from a specified Web site, may be executed by the connected computer **3** within this session. While the first session is still active, the user **2** may decide to open a second application. The second application may also

reside on the connected computer **3**, may be local to the user terminal **1** (e.g., a desktop application), or may be shared between the two. For example, the second application may be the downloading of content associated with a second Web page from a different specified Web site. The user terminal **1** may detect when the user **2** requests that the second application be launched, for example, by detecting when the user activates a link embedded in a Web page, activates a screen icon, enters an application execution command, accesses an executable computer file, or the like. In another example, the second application may be a desktop application such as a word processing, spreadsheet, or printer driver application. In an embodiment of the invention, the second application may reside on a second connected computer that may communicate with the user terminal **1** over the communication network **4**.

A timeout function associated with the first application may include performing the steps shown in the flowchart in FIGURE 2. Block **101** shows execution of the application being started. This step may be performed by the application host, i.e., either the user terminal **1**, the connected computer **3** or, for example, in the case of a distributed application, a combination of the two. At block **102**, a timeout clock associated with the application may be set (where execution of the application is just beginning) or reset to an initial state to begin counting time. The timeout clock may count up, i.e., starting from an initial state of zero and periodically incrementing until the timeout period value is reached, or count down, i.e., starting from an initial state of the countdown period value and periodically decrementing until zero is reached. As shown in block **103**, the application proceeds with normal execution of the application. This may include receiving input by, for example, causing the host to poll an input device or devices associated with the host. Alternatively, an input device or devices may create and transmit an interrupt signal when input is received. In an embodiment in which the application host is simultaneously executing multiple applications, the input may include a header identifying with which application the input is meant to be used.

In block **104**, the application determines whether the input it has received is of a type meant to prevent timeout of the application, i.e., a "keep-alive input". The type of input

necessary to keep the application alive may vary according to the application or may be selected by the user **2** or a system or network administrator. Possible keep-alive input may include simple periodic pings or be limited to specific input data messages. If a keep-alive input is received, the application returns to block **102** and resets the timeout clock to its initial state.

If a keep-alive input is not received, the application checks whether the timeout clock has exceeded a specified timeout period. The timeout period may be set by default or specified by the user **2**, or a system or network administrator. If the timeout clock does not exceed the timeout period, the application returns to block **103** and continues to execute the application. However, if the timeout clock exceeds the timeout period value, the application terminates, as shown in block **106**. In particular embodiments, termination block **106** may include steps for sending the user **2** a warning message that the application is about to be timed out. In these embodiments, the user **2** may be allowed to send a signal to the application requesting that the application not be timed out. In effect, in such an embodiment, block **106** may include sub-blocks similar to blocks **103**, **104**, and **105**.

FIGURE 3 depicts a flowchart describing the operation of an embodiment of the present invention in which a keep-alive input may be sent to prevent timeout of the first application. According to this embodiment, there may be an application (the "keep-alive function") residing on a user terminal **1** or a connected computer **3** which is initiated (as shown in block **203**) if the user has caused a first application to be launched (as shown in block **201**) and subsequently, prior to termination of the first application, causes a second application to be launched (as shown in block **202**). The keep-alive function need not reside on the same user terminal **1** or connected computer **3** as either the first or second application. In an embodiment of the invention, the user **2** may be asked whether the keep-alive process should be initiated and the keep-alive may be initiated only if the user **2** responds affirmatively to this inquiry.

In an embodiment of the invention, upon initiation, the keep-alive process may collect information about the timeout function associated with the first software application in block **204**. This may be done by transmitting a request for such information to the application host for

the first application, i.e., the connected computer **3**. In a client-server session-based embodiment of the invention, the request may include the session identifier. In embodiments of the invention, the user terminal may access information identifying particular environment variables or parameters associated with the timeout function ("timeout parameters") for the first application and may include in the request the identity of the timeout parameters in the request. Alternatively, the request may be general and may be answered with information related to all environment variables and parameters. In such an embodiment, the keep-alive function may sort through the environment variable/parameter information to identify timeout parameters (if precisely known) or likely timeout parameters. Likely timeout parameters may include all parameters with the term "timeout" or some variation thereof in the parameter name. Likely timeout parameters may also be limited by the value type associated with the parameter.

For example, in an embodiment of the invention, the parameter for the duration of the timeout clock may be sought. However the precise name of the parameter associated with the duration of the timeout clock for the first application may not be known. The request transmitted to the connected computer **3** may yield a general response including all environment variables and/or parameters. From the returned environment variable/parameter information, the keep-alive function may identify only parameters with values that are numerical values as likely timeout parameters, since the value of the parameter associated with the duration of the timeout clock is likely to be a numerical value (such as an integer number equivalent to the number of seconds the timeout clock is supposed to count). Where multiple likely timeout parameters are identified, the keep-alive function may choose one or more parameters from the identified likely timeout parameters based on the values of the likely timeout parameters. For example, the first application may have separate environment variables/parameters associated with the application timeout clock, a message acknowledgement receipt timeout clock, etc., all of which have the term "timeout" in their names and all of which have numerical values. In such a situation, the keep-alive function may conservatively select the likely timeout parameter having the smallest value as the value to use in estimating when the first application will expire.

In alternative embodiments, the user terminal **1** may access information precisely identifying the timeout parameter needed. In such a case, the request may specifically identify the timeout parameter sought. Timeout parameter identification information may include lists of names for timeout parameters used by functions that the user terminal **1** is capable of launching, timeout parameters commonly used in applications, and the like.

Timeout function information sought by the keep-alive function may include the default, user-specified and/or system or network administrator-specified timeout period for the first application, a current application timeout clock value, and the type of keep-alive input that would cause the application timeout clock to be reset (for example, a specified input format or content). Based on this information, the keep-alive function may set its own function timeout clock (the "process clock") to an initial state, as shown in block **205**. The function timeout clock may be synchronized with the application timeout clock. The value of the function timeout clock may be set so that the function timeout clock will expire before the application timeout clock expires. As with the application timeout clock, the function timeout clock may be set to an initial state of zero and be incremented or the process clock may be set to an initial state equal to the timeout period value and be decremented. Alternatively, the keep-alive function may rely solely on the application timeout clock. However, this type of embodiment may suffer from communication delays in systems involving distributed applications and/or a long-distance or shared communication network **4**.

In some embodiments of the invention, the keep-alive process may not gather first application timeout function information. The keep-alive process may be prevented from collecting such information due to security measures implemented by the connected computer **3** or the amount or type of available information may be difficult to collect or process. In these embodiments of the invention, the keep-alive function may set a function timeout clock to a duration known or believed to be shorter than the timeout process clock.

In an embodiment of the invention, the keep-alive process may use collected timeout information related to keep-alive input type to create a customized or application-specific, keep-

alive input. For example, if the first application is an application for downloading Web page content, the keep-alive input may contain instructions to "refresh" the content.

As shown by block **206** and **207**, when the process clock nears the timeout period value if the clock is being incremented or zero if the clock is being decremented, the keep-alive process transmits a keep-alive input to the first application. The keep-alive process may create the keep-alive input based upon the application timeout information collected in the step shown in block **204**. The process clock value at which a keep-alive input will be transmitted may be set by the process so as to compensate for synchronization errors between the process clock and the application timeout clock.

In embodiments of the invention in which the keep-alive process maintains an independent function timeout clock, the function timeout clock may be reset each time a keep-alive input is transmitted to the application or, as shown in block **208**, only after it has been verified that the application timeout clock has been reset. In other words, some embodiments of the invention may omit block **208** and automatically reset the function timeout clock upon transmission of the keep-alive input. In order to verify that the application timeout clock has been reset, the keep-alive function may send a request for timeout parameter information to the connected computer **3** as part of the keep-alive input or in a separate transmission.

FIG. 4 is a flowchart depicting the process that may be followed by an alternative embodiment of the invention. Unlike the embodiment depicted in FIG. 3, in this embodiment of the invention, the keep-alive function does not cause a keep-alive input to be sent to the first application host. Instead, the keep-alive function checks the status of the first application, as shown in block **307**, and informs the user of the status of the first application, as shown in block **308**. The keep-alive function may check the status of the first application by sending a status query message to the first application host. In a client-server embodiment of the invention, the status query message may include a session and/or application identifier. In block **309**, the keep-alive function may then decide whether to reset the function timeout clock based on whether the first application is still alive. The frequency at which status query messages are sent to the first

application host may be set so as to be shorter than the duration of the timeout clock for most applications or for all applications that the user terminal **1** may be capable of causing to launch. Status query messages may be transmitted when the keep-alive function timeout clock has reached a specified state (e.g., a particular time value remaining). The keep-alive function timeout clock may be independent from a first application timeout clock that may be maintained by the first application host. In block **309**, the keep-alive function may decide whether to reset the keep-alive function timeout clock based upon the status of the application, e.g., if the application is still alive. In alternative embodiments of the invention, if the application is found to have been terminated, the keep-alive function may terminate itself. In an embodiment of the invention, the status query message may include a request for first application timeout function information.

While the description above refers to particular embodiments of the present invention, it should be readily apparent to people of ordinary skill in the art that a number of modifications may be made without departing from the spirit thereof. The accompanying claims are intended to cover such modifications as would fall within the true spirit and scope of the invention. The presently disclosed embodiments are, therefore, to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than the foregoing description. All changes that come within the meaning of and range of equivalency of the claims are intended to be embraced therein.